Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

PARALLEL BAYESIAN POLICIES FOR FINITE-HORIZON MULTIPLE COMPARISONS WITH A KNOWN STANDARD

Weici Hu
Peter I. Frazier

School of Operations Research & Information Engineering
Cornell University
206 Rhodes Hall
Ithaca, NY 14853, USA

Jing Xie

American Express Company
200 Vesey St
New York, NY 10285, USA

ABSTRACT

We consider the problem of multiple comparisons with a known standard, in which we wish to allocate simulation effort efficiently across a finite number of simulated systems, to determine which systems have mean performance exceeding a known threshold. We suppose that parallel computing resources are available, and that we are given a fixed simulation budget. We consider this problem in a Bayesian setting, and formulate it as a stochastic dynamic program. For simplicity, we focus on Bernoulli sampling, with a linear loss function. Using links to restless multi-armed bandits, we provide a computationally tractable upper bound on the value of the Bayes-optimal policy, and an index policy motivated by these upper bounds.

1 INTRODUCTION

Multiple comparisons with a known standard (MCS) is a widely considered problem in simulation. In this problem, one wishes to use simulation to determine, for each among a finite pool of simulatable systems, which ones have an expected output measure that exceeds a known threshold. In this paper, we use Bayesian statistics and dynamic programming to study how one should allocate simulate effort in the MCS problem, so as to best support this final determination.

The MCS problem arises in at least two distinct ways in simulation applications. First, it arises when determining which options perform better than some standard option whose performance is so well-estimated that it can be treated as known (Nelson and Goldsman 2001). Second, it arises when determining which options have a secondary performance measure that satisfies a constraint (Andradóttir and Kim 2010). The MCS problem also arises outside of simulation, in crowdsourcing service centers like Amazon's Mechanical Turk, when allocating a budget across workers who label items (e.g., images, documents), to best support accurate classification of these items (Chen, Lin, and Zhou 2013).

We consider a variant of the MCS problem in which parallel computing resources are available. The growing availability of parallel computing resources presents new opportunities to perform simulation analysis at larger scales, but also imposes constraints on the way simulation effort is allocated. In our model, simulation effort is allocated batch-sequentially: simulations are performed in batches, and we decide how many additional replications to perform for each system at the start of each batch based on the results of previous batches. We are given a fixed budget, specified as a number of parallel computing resources and a number of batches, and our goal is to allocate these batches of simulation efficiently, so as to best allow correct classification of the systems once our simulation budget is exhausted.

We formulate the MCS problem in a Bayesian framework, and we measure the performance of a batch-sequential procedure by its average case performance, averaging across problem instances drawn from the prior and across simulation noise. While the Bayes-optimal procedure is characterized by the dynamic programming equations (Frazier 2011), the curse of dimensionality makes solving this dynamic program computationally intractable for problems with many systems.

Rather than solving this dynamic program exactly, we provide a computationally tractable upper bound on its value. This allows us to evaluate the quality of sub-optimal heuristic policies relative to this upper bound. This provides guidance to the development and improvement of heuristic policies, in the form of information about the optimality gap. The analysis technique used in this upper bound is a Lagrangian relaxation on the total number of simulations performed in any given batch. Using the Lagrange multipliers obtained from this relaxation, we also develop a heuristic policy, and use numerical experiments to demonstrate that it performs close to the upper bound on optimal in the problem setting studied.

This paper builds on the previous work Xie and Frazier (2013b), which also considered the Bayesian MCS problem. That paper considered the sequential setting, without parallel resources, and provided a computationally efficient method for computing the Bayes-optimal policy under two assumptions about limitations of sampling: that there is a time horizon that is random and exponentially distributed; or there is no time horizon, and we pay a fixed cost for each sample. Our current work differs from that work by considering parallelism, and by considering a fixed budget. While the infinite-horizon fixed-cost-per-sample model in Xie and Frazier (2013b) is quite natural for cloud computing settings, and the random exponentially distributed horizon is attractive for its computational tractability, using a fixed horizon is more natural than either model in Xie and Frazier (2013b) when allocating computing resources that are owned rather than rented. While we focus on the parallel setting and Xie and Frazier (2013b) focused on the sequential setting, our work can also provide an upper bound for the sequential setting with fixed horizon by setting the number of parallel nodes to 1.

For simplicity in this paper, we consider only Bernoulli samples, with a linear loss function. However, the techniques developed in this paper should also be adaptable to other parametric sampling distributions with conjugate priors, and other loss functions.

Our model assumes synchronous computations, in which we wait for all simulations in a batch to complete before starting the next batch. This approach is reasonable when the variability in the time to simulate a system is small enough to allow waiting until simulations finish before starting the next batch. Such assumptions are more commonly met in controlled high-performance computing environments, and are less common in cloud computing environments. If computation time is highly variable, it may be more appropriate to model computation as asynchronous.

Our Lagrangian relaxation of a budget constraint in the MCS problem is related to Lagrangian relaxations in restless multi-armed bandit problems (Whittle 1988, Gittins, Glazebrook, and Weber 2011). Indeed, if one added an additional constraint that each system can be simulated at most once in any batch, then our MCS problem can be reformulated as a restless multi-armed bandit problem where we can pull multiple arms in each round: in this restless multi-armed bandit problem, each system is an arm, pulling the arm corresponds to simulating that system; the reward from this pull is the improvement in our ability to classify this system; and the number of arms we can pull in a round is the number of parallel resources. The restless multi-armed bandit framework of Whittle (1988) is most well-known for allowing bandit arms to evolve when they are not pulled, but also allows multiple pulls per round. Our analysis can be seen as examining a generalization of restless bandits in which each arm can be pulled multiple times per round, and our heuristic policy can be seen as a generalization of the Whittle index policy to this setting.

Our use of a Lagrangian relaxation to study the Bayesian formulation of the MCS problem is also similar to Xie and Frazier (2013a), which used a Lagrangian relaxation to bound the value of the Bayes-optimal procedure for the ranking and selection problem.

While we consider the MCS problem in the Bayesian setting, much of the previous work on the MCS problem has considered non-Bayesian settings. This previous work includes the one-stage procedures by Paulson (1952), Dunnett (1955), the two-stage procedures by Dudewicz and Dalal (1983), Bofinger and Lewis (1992), Damerdji and Nakayama (1996), and work on indifference-zone ranking and selection (see the survey Kim and Nelson (2007)). The more recent frequentist work includes Batur and Kim (2010) which provides a fully sequential procedure under stochastic constraint, and Healey, Andradóttir, and Kim (2014) which further allows correlation across systems.

In the version of the MCS problem that we consider, we emphasize that our standard has known value, and we seek to determine only whether each system is better or worse than this standard. We do not consider standards with unknown value, produce joint confidence intervals, nor select the best among those systems performing better than standard. This is in contrast with much previous work on multiple comparisons (Nelson and Goldsman 2001, Kim 2005). The variant of the MCS problem that we consider has also been called *feasibility determination* (Szechtman and Yücesan 2008).

This paper is organized as follows. In Section 2 we formally state the problem. In Section 3, we provide a computationally tractable upper bound on the value of a Bayes-optimal procedure, and a method for computing this upper bound. In Section 4 we present a heuristic motivated by this upper bound, which is similar to the Whittle index policy for restless multi-armed bandits. In Section 5 we present numerical results in which we demonstrate that the Whittle index policy performs close to an optimal policy. Lastly we present our numerical results and conclusion in Sections 5 and 6.

2 PROBLEM FORMULATION

We have k systems, each of which can be simulated using a stochastic simulation. When we simulate system $x \in \{1,\dots,k\}$, we observe a Bernoulli(θ_x) random variable, indicating the system's performance in that simulation. We will think of an outcome of 1 as indicating the system succeeded in that simulation, and 0 as indicating failure. Although we expect that the methods we develop in this paper can be extended to simulations that generate non-Bernoulli samples, we focus on the Bernoulli setting here for simplicity. The sampling means θ_x are initially unknown, and we wish to determine through simulation, for each system x, whether θ_x is greater than some known threshold d_x. This threshold may differ across systems.

We adopt a Bayesian formulation, in which we seek to do well on average with respect to a prior probability distribution over the unknown sampling means $\theta_1,\dots,\theta_k$. We consider Bayesian prior probability distributions under which

$$\theta_x \sim \text{Beta}(\alpha_{0,x},\beta_{0,x}),$$

with independence across x, for some given values $\alpha_{0,x},\beta_{0,x}$. We assume a Beta prior for tractability: the Beta prior is conjugate to the Bernoulli likelihood (DeGroot 1970), and so this assumption allows our posterior distribution to remain Beta-distributed.

We perform N batches of simulations, performing at most m simulations in parallel in each batch. At the start of each batch $n = 1,\dots,N$, we choose the number of samples $z_{n,x}$ to take from each system x, making sure to satisfy the constraint $\sum_x z_{n,x} \le m$. This choice of $z_{n,x}$ may depend upon the results observed from all previous batches. We then observe the number of successes from each system x, which are conditionally binomial,

$$Y_{n,x}|\theta_x, z_{n,x} \sim \text{Binomial}(z_{n,x},\theta_x).$$

We assume conditional independence of $Y_{n,x}$ across x and from all previous samples, given θ_x and $z_{n,x}$. This precludes the use of common random numbers, but is satisfied when using independent sampling. After observing this $Y_{n,x}$, we update our posterior distribution on θ_x to obtain (DeGroot 1970)

$$\theta_x|z_{1,x},Y_{1,x},\dots,z_{n,x},Y_{n,x} \sim \text{Beta}(\alpha_{n,x},\beta_{n,x}),$$

where $\alpha_{n,x} = \alpha_{0,x} + \sum_{n' \le n} Y_{n',x}$ can be interpreted as the effective number of successes from system x, and $\beta_{n,x} = \beta_{0,x} + \sum_{n' \le n}(z_{n',x} - Y_{n',x})$ as the effective number of failures. This posterior is independent across x.

We use dynamic programming to analyze this problem. To support this, for each x and n, we define the state variable $S_{n,x} = (\alpha_{n,x},\beta_{n,x})$. To streamline the discussion, we define $\boldsymbol{\alpha}_n = (\alpha_{n,1},...,\alpha_{n,k})$, $\boldsymbol{\beta}_n = (\beta_{n,1},..,\beta_{n,k})$, $\mathbf{S}_n = (S_{n,1},...,S_{n,k})$, and $\mathbf{z}_n = (z_{n,1},\dots,z_{n,k})$. Let Λ_n be the space in which $\mathbf{S}_n$ takes values, $\Lambda_n = \big\{(\alpha_{0,1}+s_1,\beta_{0,1}+f_1,\dots,\alpha_{0,k}+s_k,\beta_{0,k}+f_k) : s_x, f_x \in \mathbb{Z}_+ \forall x,\ \sum_{x=1}^k s_x + f_x \le mn\big\}$.

We stop sampling after batch N and decide, for each system x, whether to label θ_x as above or below the threshold d_x. If we label it as above, we receive a reward of $\theta_x - d_x$. Otherwise, we receive $d_x - \theta_x$. The total terminal reward is the sum of these rewards across the systems. As shown in Xie and Frazier (2013b), to maximize the conditional expected value of this reward given what we know after our simulations are complete (which is summarized in $S_{N,x}$), we should choose to receive $\theta_x - d$ whenever the conditional expected value of this reward $E\left[\theta_x - d_x | S_{N,x}\right]$ is positive, and to receive $d_x - \theta_x$ when it is negative. When making decisions in this way, the conditional expected terminal reward received is

$$\max\left\{E\left[\theta_x - d_x|S_{N,x}\right], E\left[d_x - \theta_x|S_{N,x}\right]\right\} = \left|E\left[\theta_x - d_x|S_{N,x}\right]\right| = \left|\frac{\alpha_{N,x}}{\alpha_{N,x}+\beta_{N,x}} - d_x\right|.$$

Summing this reward across systems x, our conditional expected terminal reward is

$$r(\mathbf{S}_N) = \sum_{x=1}^{k}\left|\frac{\alpha_{N,x}}{\alpha_{N,x}+\beta_{N,x}} - d_x\right|.$$

We optionally allow a cost $c_x \geq 0$ for each sample generated for system x, in the same units as the objective function. For example, if the reward is monetary, then the cost might be the payment made to a cloud computing service such as Amazon Ec2 or Microsoft Azure for the computer time required to run a simulation. If the computers are owned, rather than rented, then it may be appropriate to set $c_x = 0$. Combining this optional sampling cost with the reward $r(\mathbf{S}_N)$ gives the conditional expected overall reward

$$r(\mathbf{S}_N) - \sum_{x=1}^{k}\sum_{n=1}^{N} c_x z_{n,x}.$$

Our goal is to find an algorithm or *policy* for choosing the samples to take, $z_{n,x}$, so as to maximize the expected value of this reward. A policy is a rule for choosing how to allocate the next batch of samples, based on the results of the previous samples as summarized by $\mathbf{S}_n$. Formally, a policy π is a sequence of mappings $\pi = (\pi_0, \ldots, \pi_{N-1})$, where $\pi_n : \Lambda_n \mapsto \mathbb{Z}_+^k$ maps the state $\mathbf{S}_n$ to the action $\mathbf{z}_n = \pi_n(\mathbf{S}_n)$, while satisfying the constraint on the number of samples in each batch. The set of all policies is $\Pi = \left\{\pi = (\pi_0, \ldots, \pi_{N-1}) : \sum_{x=1}^{k} \pi_{n,x}(\mathbf{S}) \leq m \ \forall n = 0, \ldots, N-1, \mathbf{S} \in \Lambda_n\right\}$, where $\pi_{nx}(\mathbf{S})$ indicate the xth component of $\pi_n(\mathbf{S})$.

Each policy π induces a probability distribution over $(\mathbf{S}_0, \mathbf{z}_0, \ldots, \mathbf{S}_{N-1}, \mathbf{z}_{N-1}, \mathbf{S}_N)$, which we call P^π. We let E^π indicate the expectation taken with respect to this probability distribution. We define,

$$V_n^\pi(\mathbf{S}) = E^\pi\left[r(\mathbf{S}_N) - \sum_{x=1}^{k}\sum_{n'=n+1}^{N} c_x z_{n',x} \mid \mathbf{S}_n = \mathbf{S}\right],$$

for $0 \leq n \leq N$, which is the conditional expectation of the future reward under policy π, starting from state $\mathbf{S}$ at time n. We also define the value function, used below in our dynamic programming approach, as

$$V_n(\mathbf{S}) = \sup_{\pi\in\Pi} V_n^\pi(\mathbf{S}).$$

The (unconditional) expected reward obtained under π is $V_0^\pi(\mathbf{S}_0)$, and an optimal policy π^* is any for which $\pi^* \in \operatorname{argmax}_{\pi\in\Pi} V_0^\pi(\mathbf{S}_0)$.

The problem $\sup_{\pi\in\Pi} V_0^\pi(\mathbf{S}_0)$ is a Markov decision process with finite horizon and finite state space, and its solution is characterized by the dynamic programming equations. To apply dynamic programming, we first write down the dynamic programming equation, which is a recursive relation for V_n:

$$V_n(\mathbf{S}_n) = \max_{\mathbf{z}_{n+1}:\sum_x z_{n+1,x}\leq m}\left\{-\sum_x c_x z_{n+1,x} + \mathbb{E}[V_{n+1}(\mathbf{S}_{n+1})|\mathbf{S}_n, \mathbf{z}_{n+1}]\right\}, \quad 0 \leq n \leq N-1. \tag{1a}$$

$$V_N(\mathbf{S}_N) = \sum_{x=1}^{k} \left| \frac{\alpha_{N,x}}{\alpha_{N,x}+\beta_{N,x}} - d_x \right|. \tag{1b}$$

Any policy whose actions achieve the maximum in (1a) is optimal (Dynkin and Yushkevich 1979).

We can use these recursive equations to compute V_n directly, first calculating $V_N(\mathbf{S}_N)$ for all possible $\mathbf{S}_N \in \Lambda_N$, and then proceeding in a backward recursion, using previously computed values of $V_{n+1}(\mathbf{S}_{n+1})$ to calculate $V_n(\mathbf{S}_n)$ for all $\mathbf{S}_n \in \Lambda_n$. Then, given these computed value functions, we can compute an optimal policy.

While this direct dynamic programming approach is theoretically well understand, it quickly becomes computationally intractable as k grows. This is because the state space at time n, Λ_n, has $O((mn)^{2k})$ elements, and so storing the value function at all possible states has a memory requirement that scales exponentially in k. Computation also scales exponentially in k. This computational infeasibility due to the large dimension of the problem is generally referred to as the "curse of dimensionality" (Powell 2007).

The computationally intractability of computing an optimal policy when k is large leads us to consider other characterizations that can be computed more easily. In the next section, we show how to compute an upper bound on the value of the optimal policy that scales linearly in k, rather than exponentially.

3 UPPER BOUND

In this section, we provide a computationally tractable upper bound on the value of an optimal policy. This bound can be used to calculate an optimality gap for any desired heuristic policy π, by comparing the upper bound to an estimate of the heuristic policy's value $V^{\pi}(\mathbf{S}_0)$ obtained from direct simulation. This in turn can be used to judge whether a particular heuristic policy is good enough to be used in practice, or if more development (either of better heuristics or tighter upper bounds) would be worthwhile.

The main idea in our upper bound is to relax the constraints $\sum_{x=1}^{k} z_{n,x} \leq m$ with a Lagrange multiplier. As the first step, we introduce values $\boldsymbol{\lambda} = (\lambda_1,\ldots,\lambda_N) \in \mathbb{R}_+^k$. The value λ_n will be the Lagrange multiplier for the constraint $\sum_{x=1}^{k} z_{n,x} \leq m$. For each $\boldsymbol{\lambda}$, define a modified value function $V_n^{\boldsymbol{\lambda}}(\mathbf{S}_n)$ via the following recursion:

$$V_n^{\boldsymbol{\lambda}}(\mathbf{S}_n) = \max_{\mathbf{z}_{n+1} \in \{0,1,\ldots,m\}^k} \left\{ -\sum_x c_x z_{n+1,x} + \mathbb{E}\left[V_{n+1}^{\boldsymbol{\lambda}}(\mathbf{S}_{n+1}) | \mathbf{S}_n, \mathbf{z}_{n+1} \right] - \lambda_{n+1} \left(\sum_{x=1}^{k} z_{n+1,x} - m \right) \right\}, n \leq N-1, \tag{2a}$$

$$V_N^{\boldsymbol{\lambda}}(\mathbf{S}_N) = \sum_{x=1}^{k} \left| \frac{\alpha_{N,x}}{\alpha_{N,x}+\beta_{N,x}} - d_x \right|. \tag{2b}$$

We will see below in Lemma 2 that $V_0^{\boldsymbol{\lambda}}(\mathbf{S}_0)$ provides an upper bound on the value function $V_0(\mathbf{S}_0)$, and hence on the value of an optimal policy.

While computing $V_0^{\boldsymbol{\lambda}}(\mathbf{S}_0)$ directly using the recursion (2) would also seem to require storing a value for every state in the state space, just as the recursion (1), we will see below in Lemma 1 that it can be computed instead as the sum of other modified value functions, each of which corresponds to a single system, and which can be computed by considering a much smaller state space whose size does not grow with k. This will allow efficient computation.

Toward this end, we define the function $V_{n,x}^{\boldsymbol{\lambda}}$ for each x via the recursive relation

$$V_{n,x}^{\boldsymbol{\lambda}}(S_{n,x}) = \max_{z_{n+1,x} \in \{0,\ldots,m\}} \left\{ -z_{n+1,x}(c_x + \lambda_{n+1}) + \mathbb{E}\left[V_{n+1,x}^{\lambda}(S_{n+1,x}) \middle| S_{n,x}, z_{n+1,x} \right] \right\},\ n \leq N-1, \tag{3a}$$

$$V_{N,x}^{\boldsymbol{\lambda}}(S_{N,x}) = \left| \frac{\alpha_{N,x}}{\alpha_{N,x}+\beta_{N,x}} - d_x \right|. \tag{3b}$$

$V_{n,x}^{\boldsymbol{\lambda}}(S_{n,x})$ is the value function for a dynamic program corresponding to an MCS problem with a single system x, with a new sampling cost $c_x + \lambda_{n+1}$ for samples in batch n. The cost of sampling depends on the batch n. The additional cost λ_{n+1} beyond the cost c_x in our original model arises from the way in which we relaxed the constraint on the number of samples in each batch in the multi-system problem. $V^{\boldsymbol{\lambda}}_{n,x}(S_{n,x})$ can be computed directly using this recursive relation, because the space $\Lambda_{n,x} = \{(\alpha_{0,x}+s_x, \beta_{0,x}+f_x) : s_x, f_x \in \mathbb{Z}_+, s_x+f_x \leq mn\}$ in which $S_{n,x}$ takes values has only $O(nm)$ elements, and does not grow exponentially in the problem parameters.

The following lemma shows that $V^{\boldsymbol{\lambda}}_n(\mathbf{S}_n)$ can be computed directly from $V^{\boldsymbol{\lambda}}_{n,x}(S_{n,x})$, allowing its efficient computation.

Lemma 1 For any $\boldsymbol{\lambda} \geq \mathbf{0}$,

$$V^{\boldsymbol{\lambda}}_n(\mathbf{S}_n) = \sum_{x=1}^{k} V^{\boldsymbol{\lambda}}_{n,x}(S_{n,x}) + m \sum_{n'=n+1}^{N} \lambda_{n'}, \tag{4}$$

Proof. This proof uses an inductive argument. At time N, (4) holds from (3b) and (2b). Assume (4) holds for time $n+1$. Then,

$$V^{\boldsymbol{\lambda}}_n(\mathbf{S}_n) = \max_{\mathbf{z}_{n+1}\in\{1,..,m\}^k} \Big\{ -\sum_{x=1}^{k} c_x z_{n+1,x} + \mathbb{E}\Big[\sum_{x=1}^{k} V^{\boldsymbol{\lambda}}_{n+1,x}(S_{n+1,x}) + m\sum_{n'=n+2}^{N}\lambda_{n'} \Big| \mathbf{S}_n, \mathbf{z}_{n+1}\Big] - \lambda_{n+1}(\sum_{x=1}^{k} z_{n+1,x} - m)\Big\} \tag{5}$$

$$= \max_{\mathbf{z}_{n+1}\in\{1,...,m\}^k} \Big\{ -\sum_{x=1}^{k}\Big(c_x z_{n+1,x} + \mathbb{E}\Big[V^{\boldsymbol{\lambda}}_{n+1,x}(S_{n+1,x})\Big| S_{n,x}, z_{n+1,x}\Big]\Big) - \lambda_{n+1}\sum_{x=1}^{k} z_{n+1,x}\Big\} + m\sum_{n'=n+1}^{N}\lambda_{n'} \tag{6}$$

$$= \sum_{x=1}^{k} \max_{z_{n+1,x}\in\{1,...,m\}} \Big\{ -c_x z_{n+1,x} + \mathbb{E}\Big[V^{\boldsymbol{\lambda}}_{n+1,x}(S_{n,x})\Big| S_{n+1,x}, z_{n+1,x}\Big] - \lambda_{n+1,x} z_{n+1,x}\Big\} + m\sum_{n'=n+1}^{N}\lambda_{n'} \tag{7}$$

$$= \sum_{x=1}^{k} V^{\boldsymbol{\lambda}}_{n,x}(S_{n,x}) + m\sum_{n'=n+1}^{N}\lambda_{n'}. \tag{8}$$

Equation (5) follows from the inductive hypothesis. Equality (6) holds due to the fact that each system is independent from the current action and states of the other systems. Equality (7) holds because the x^{th} summand in the summation of (6) only depends on $z_{n+1,x}$; to maximize the sum is to maximize each of the summands by choosing the right $z_{n+1,x}$. □

Setting $n = 0$ in the above lemma, we obtain a readily computed expression for $V^{\boldsymbol{\lambda}}_0(\mathbf{S}_0)$,

$$V^{\boldsymbol{\lambda}}_0(\mathbf{S}_0) = \sum_{x=1}^{k} V^{\boldsymbol{\lambda}}_{0,x}(S_{0,x}) + m\sum_{n=1}^{N}\lambda_n. \tag{9}$$

The following lemma shows that this is an upper bound on the value function for our original MCS problem.

Lemma 2 For any $\boldsymbol{\lambda} \geq \mathbf{0}$,

$$V^{\boldsymbol{\lambda}}_0(\mathbf{S}_0) \geq V_0(\mathbf{S}_0). \tag{10}$$

Proof. We first prove

$$V^{\boldsymbol{\lambda}}_n(\mathbf{S}_n) \geq V_n(\mathbf{S}_n), \tag{11}$$

for all n such that $0 \leq n \leq N$ using an inductive argument. Then (10) holds automatically. At time N, inequality (11) follows from equation (1b) and (2b). Assume (11) holds at time $n+1$. Noticing

$\sum_{x=1}^{k} z_{n+1,x} - m \leq 0$ as stated in the problem formulation, we have

$$\begin{aligned}
V_n^{\boldsymbol{\lambda}}(\mathbf{S}_n) &= \max_{\mathbf{z}_{n+1} \in \{1,..,m\}^k} \{-\sum_{x=1}^{k} c_x z_{n+1,x} + \mathbb{E}[V_{n+1}^{\boldsymbol{\lambda}}(\mathbf{S}_{n+1}) | \mathbf{S}_n, \mathbf{z}_{n+1}] - \lambda_{n+1}(\sum_{x=1}^{k} z_{n+1,x} - m)\} \\
&\geq \max_{\mathbf{z}_{n+1} \in \{1,..,m\}^k} \{-\sum_{x=1}^{k} c_x z_{n+1,x} + \mathbb{E}[V_{n+1}^{\boldsymbol{\lambda}}(\mathbf{S}_{n+1}) | \mathbf{S}_n, \mathbf{z}_{n+1}]\} \\
&\geq \max_{\mathbf{z}_{n+1} \in \{1,..,m\}^k} \{-\sum_{x=1}^{k} c_x z_{n+1,x} + \mathbb{E}[V_{n+1}(\mathbf{S}_{n+1}) | \mathbf{S}_n, \mathbf{z}_{n+1}]\} \\
&\geq \max_{\mathbf{z}_{n+1} \in \{1,..,m\}^k, \text{s.t. } \sum_{x=1}^{k} z_{n+1,x} \leq m} \{-\sum_{x=1}^{k} c_x z_{n+1,x} + \mathbb{E}[V_{n+1}(\mathbf{S}_{n+1}) | \mathbf{S}_n, \mathbf{z}_{n+1}]\} = V_n(\mathbf{S}_n). \qquad \square
\end{aligned}$$

Now we have that for any $\boldsymbol{\lambda} \geq \mathbf{0}$, $V_0^{\boldsymbol{\lambda}}(\mathbf{S}_0)$ forms an upper bound on the optimal total expected reward of the original problem. We then obtain the tightest upper bound of this form by selecting the infimum.

Theorem 1

$$\mathrm{UB}(\mathbf{S}_0) = \inf_{\boldsymbol{\lambda} \geq \mathbf{0}} \Big[\sum_{x=1}^{k} V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x}) + m \sum_{n=1}^{N} \lambda_n \Big] \tag{12}$$

gives an upper bound on $V_0(\mathbf{S}_0)$.

Proof. This result follows directly from (9) and Lemma 2. $\square$

While we have argued that $V_0^{\boldsymbol{\lambda}}(\mathbf{S}_0)$ can be computed efficiently as the sum of values from small dynamic programs, each corresponding to a single system. We now show how the infimum in $\mathrm{UB}(\mathbf{S}_0)$ can be computed. Define $B(\mathbf{S}_0, \boldsymbol{\lambda}) = \sum_{x=1}^{k} V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x}) + m \sum_{n=1}^{N} \lambda_n$. To compute the upper bound in equation (12), we first show that $B(\mathbf{S}_0, \boldsymbol{\lambda})$ is convex in $\boldsymbol{\lambda}$ and subsequently (12) is a convex optimization problem. Moreover, it is possible to compute the subgradient of $\boldsymbol{\lambda} \mapsto B(\mathbf{S}_0, \boldsymbol{\lambda})$, allowing the use of a first-order convex optimization method.

Lemma 3 $B(\mathbf{S}_0, \boldsymbol{\lambda})$ is convex in $\boldsymbol{\lambda}$ for any $\boldsymbol{\lambda} \geq \mathbf{0}$.

Proof. Since $m \sum_{n=1}^{N} \lambda_n$ is convex in $\boldsymbol{\lambda}$, it is sufficient to show $V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x})$ is convex in $\boldsymbol{\lambda}$ for all x. This shall be shown by induction. At time N, $V_{N,x}^{\boldsymbol{\lambda}}(S_{N,x}) = |\frac{\alpha_{N,x}}{\alpha_{N,x} + \beta_{N,x}} - d_x|$ is constant thus convex in $\boldsymbol{\lambda}$. Assume the result holds true for $V_{n+1}^{\boldsymbol{\lambda}}(S_{n+1,x})$, then same is true for $\mathbb{E}[V_{n+1}^{\boldsymbol{\lambda}}(S_{n+1,x}) | S_{n,x}, z_{n+1,x}]$ for any fixed $S_{n,x}$ and $z_{n+1,x}$. $-z_{n+1,x}(c_x + \lambda_{n+1})$, which is linear in λ_{n+1}, is also convex in $\boldsymbol{\lambda}$. Since the maximum of convex functions is still convex, $V_{n,x}^{\lambda}(S_{n,x}) = \max_{z_{n+1,x}} \{-z_{n+1,x}(c_x + \lambda_{n+1}) + \mathbb{E}[V_{n+1}^{\lambda}(S_{n+1,x}) | S_{n,x}, z_{n+1,x}]\}$ is convex in $\boldsymbol{\lambda}$. Subsequently we have $V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x})$ is also convex through an inductive argument. $\square$

To allow the use of a first-order convex optimization method for solving (12), which are generally faster than derivative-free methods, we much provide a method for computing the subgradient of $B(\mathbf{S}_0, \boldsymbol{\lambda})$ with respect to $\boldsymbol{\lambda}$. Since $B(\mathbf{S}_0, \boldsymbol{\lambda})$ is the sum of $V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x})$ and a term that is linear in $\boldsymbol{\lambda}$, it is sufficient to compute the subgradient $V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x})$. The next lemma provides an expression for this subgradient.

Before presenting this lemma, we first introduce some notation. Define $\pi_x^*(\boldsymbol{\lambda})$ to be an optimal policy obtained by solving the single-system MCS problem for system x, that is, the optimal policy for the dynamic program (3). Let $r_{n,x}(S_{n-1,x}, z_{n,x}; \lambda_n) = -z_{n,x}(c_x + \lambda_n)$ be the reward collected in state $S_{n-1,x}$ by taking action $z_{n,x}$ for the single-system MCS problem in (3). Let $r_{N+1,x}(S_{N,x})$ be the terminal reward. Then $V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x}) = \mathbb{E}^{\pi_x^*(\boldsymbol{\lambda})}[\sum_{n=1}^{N} r_{n,x}(S_{n-1,x}, z_{n,x}; \lambda_n) + r_{N+1,x}(S_{N,x}) | S_{0,x}]$.

Lemma 4 Fix any $\boldsymbol{\lambda}$, and any corresponding optimal policy $\pi_x^*(\boldsymbol{\lambda})$. Then, the vector

$$g(S_{0,x};\boldsymbol{\lambda}) = \left(-\mathbb{E}^{\pi_x^*(\boldsymbol{\lambda})}[z_{n,x}|S_{0,x}] : n = 1,\dots,N\right) \tag{13}$$

is a subgradient of $\boldsymbol{\lambda} \mapsto V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x})$ at $\boldsymbol{\lambda}$.

Proof. Let $\boldsymbol{\lambda}' \in \mathbb{R}_+^N$. Consider the dynamic program (3) with this value $\boldsymbol{\lambda}'$, which has value $V_{0,x}^{\boldsymbol{\lambda}'}(S_{0,x})$. Since $\pi_x^*(\boldsymbol{\lambda})$ is a feasible policy for this dynamic program, we have

$$\begin{aligned}
V_{0,x}^{\boldsymbol{\lambda}'}(S_{0,x}) &\geq \mathbb{E}^{\pi_x^*(\boldsymbol{\lambda})}\Big[\sum_{n=1}^{N} r_{n,x}(S_{n-1,x}, z_{n,x}; \lambda_n') + r_{N+1,x}(S_{N,x})|S_{0,x}\Big] \\
&= \mathbb{E}^{\pi_x^*(\boldsymbol{\lambda})}\Big[\sum_{n=1}^{N} r_{n,x}(S_{n-1,x}, z_{n,x}; \lambda_n) + r_{N+1,x}(S_{N,x})|S_{0,x}\Big] - \sum_{n=1}^{N}(\lambda_n' - \lambda_n)\mathbb{E}^{\pi_x^*(\boldsymbol{\lambda})}[z_{n,x}|S_{0,x}] \\
&= V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x}) + (\boldsymbol{\lambda}' - \boldsymbol{\lambda}) \cdot g(S_{0,x}, \boldsymbol{\lambda})
\end{aligned}$$

Thus $g(S_{0,x};\boldsymbol{\lambda})$ is a subgradient of $V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x})$. □

We can compute (13) recursively using the Markov property. Recall $\pi^*(\boldsymbol{\lambda})$ is an optimal policy for the single-system problem (3) when given $\boldsymbol{\lambda}$, and let $z_{n+1,x}^*(S_{n,x})$ be the action taken in state $S_{n,x}$ at time n as dictated by $\pi^*(\boldsymbol{\lambda})$. Let $P(s,n) = \mathbb{P}^{\pi^*(\boldsymbol{\lambda})}[S_{n,x} = s|S_{0,x}]$. We can then write the subgradient (13) as

$$-\mathbb{E}^{\pi_x^*(\boldsymbol{\lambda})}[z_{n,x}|S_{0,x} = s_x] = -\sum_{s' \in \Lambda_{n-1,x}} z_n^*(s')P(s', n-1). \tag{14}$$

$P(s,n)$ can then be computed recursively as

$$P(s,n) = \begin{cases} \mathbb{1}_{(s=S_{0,x})}, & \text{if } n = 0, \\ \mathbb{P}[S_{1,x} = s|S_{0,x}, z_1^*(S_{0,x})], & \text{if } n = 1, \\ \sum_{s' \in \Lambda_{n-1,x}} \mathbb{P}[S_{n,x} = s|S_{n-1,x} = s', z_n^*(s')] \cdot P(s', n-1), & \text{if } n > 1. \end{cases}$$

Hence we can compute (13). Finally, since $B(\mathbf{S}_0, \boldsymbol{\lambda})$ is the sum of the single-system values $V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x})$ and $m\sum_{n=1}^{N} \lambda_n$, we have the following subgradient of $B(\mathbf{S}_0, \boldsymbol{\lambda})$:

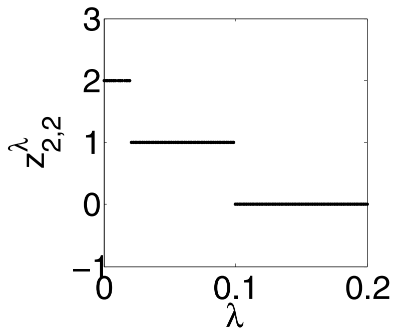
$$\sum_{x=1}^{k} \frac{\partial V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x})}{\partial \lambda_n} + m \in \partial B(\mathbf{S}_0, \boldsymbol{\lambda}) \tag{15}$$

Equation (15) allows us to compute the upper bound (12) by first-order convex optimization.

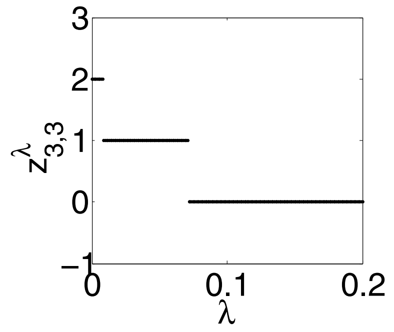
4 INDEX POLICY

In this section, we describe an index-based policy based on the same decomposition used to develop the upper bound. The intuition behind this policy is based on an unproven conjecture, but the policy is well-defined whether or not this conjecture is true. We demonstrate in numerical experiments in Section 5 that this policy performs well.

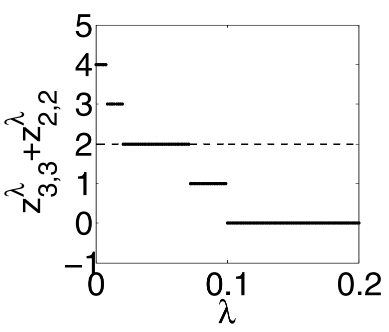
This index-based policy considers the relaxed problem (2) with a Lagrange multiplier $\boldsymbol{\lambda} = \lambda\mathbf{e}$, where $\mathbf{e} = (1,\dots,1)$ is the vector of all 1s and λ is a real number. This index-based policy is based on the intuition that, for any state $S_{n,x}$, as we increase λ and thus $\boldsymbol{\lambda}$ we should see that an optimal policy corresponding



(a)



(b)



(c)

Figure 1: This figure illustrates how $\mathbf{z}_n$ is chosen by the index-based policy with $k=2$ systems, $m=2$ parallel computing resources, and $N=20$ simulation batches. Figure (a) plots $z^{\boldsymbol{\lambda}}_{2,2}$, the optimal number of samples to take in batch 3 from system 1 when it is in state (2,2), against the value of λ; Figure (b) plots $z^{\boldsymbol{\lambda}}_{3,3}$, the optimal number of samples to take in batch 3 from system 2 when it is in state (3,3). Figure (c) plots $z^{\boldsymbol{\lambda}}_{2,2}+z^{\boldsymbol{\lambda}}_{3,3}$, the optimal total number of samples to take across both systems. The dashed line in (c) shows the constraint $m=2$, and λ^* will be the left endpoint of the solid line overlapping this dashed line. The number of samples taken from each system will be $z^{\boldsymbol{\lambda}^*}_{2,2}=1$ and $z^{\boldsymbol{\lambda}^*}_{3,3}=1$ respectively.

to $\boldsymbol{\lambda}$ should take fewer samples, because the samples are more expensive. While we conjecture that this is true, and our numerical experiments support it, we have not confirmed it theoretically.

To calculate the number of samples taken in a given state, our index-based policy varies $\boldsymbol{\lambda}$ until we find a value in which an optimal policy for the relaxed problem takes m samples (or, if no such $\boldsymbol{\lambda}$ exists, it should take as many samples as possible without taking more than m). We then sample according to the optimal policy for this $\boldsymbol{\lambda}$. When we get a new state, we repeat this process, finding a new $\boldsymbol{\lambda}$ vector, and a new sampling allocation.

We define this index-based policy more formally as follows. We first introduce some notation. Let $Q^{\boldsymbol{\lambda}}$ be the set of single-arm policies that are optimal for (3) at the given value of $\boldsymbol{\lambda}$. Let $z^{\pi}_{n,x}(S_{n-1,x})$ be the number of samples taken under a single-arm policy π at time n in state $S_{n-1,x}$.

At each time step $n=1,\ldots,N$, this policy computes $\mathbf{z}_n$ based on $\mathbf{S}_{n-1}$ in the following way:

1. Let $z^{\boldsymbol{\lambda}}_{n,x}(S_{n-1,x}) \in \left\{z^{\pi}_{n,x}(S_{n-1,x}) : \pi \in Q^{\boldsymbol{\lambda}}\right\}$ be the number of samples taken under an optimal single-arm policy with the given set of Lagrange multipliers $\boldsymbol{\lambda}$, breaking ties arbitrarily.
2. Let $\lambda^* = \inf\left\{\lambda : \sum_x z^{\boldsymbol{\lambda}}_{n,x}(S_{n-1,x}) \leq m, \boldsymbol{\lambda} = \lambda\mathbf{e}\right\}$.
3. Set $\boldsymbol{\lambda}^* = \lambda^*\mathbf{e}$.
4. Let $z_{n,x} \in \{z^{\pi}_{n,x}(S_{n-1,x}) : \pi \in Q^{\boldsymbol{\lambda}^*}\}$, so as to satisfy $\sum_{x=1}^{k} z_{n,x} \leq m$, breaking ties arbitrarily between different allocations $\mathbf{z}_n$ that satisfy this constraint.

This index-based policy leaves free the tie-breaking rule used when choosing $z^{\boldsymbol{\lambda}}_{n,x}(S_{n-1,x})$, and also when choosing $z_{n,x}$ in the final step. We conjecture that the first tie-breaking rule has no impact on the value for λ^*, although we have not confirmed this theoretically. We conjecture that the best tie-breaking rule used to choose $z_{n,x}$ in the final step is one that minimizes the number of unused cores, $m-\sum_{x=1}^{k} z_{n,x}$, and that it is always possible to find one with $m=\sum_{x=1}^{k} z_{n,x}$, but again we have not confirmed this theoretically.

Figure 1 illustrates the above procedure with two systems and two parallel resources. Figure 1a and 1b show how the optimal number of samples varies with the value of λ for each system, given their current states $(2,2)$ and $(3,3)$. Figure 1c shows the optimal total number of samples across both systems. Since there are two parallel resources, the constraint is $m = 2$. Hence in this case λ^* is the left end point of the interval at height $z^{\boldsymbol{\lambda}}_{2,2} + z^{\boldsymbol{\lambda}}_{3,3} = 2$.

5 NUMERICAL RESULTS

In this section, we present numerical results illustrating the upper bound and the index-based policy, as well as the baseline equal allocation policy.

We consider 4 different value of k: $k = 2, 4, 8, 16$. For each value of k, we set the time horizon $N = 5$, the threshold value $d_x = 0.2$ for all $x \in \{1, ..., k\}$, and the number of parallel computing resources $m = k$. We set the initial state to be the same for every system: $S_{0,x} = (1, 1)$. We first calculated the upper bounds according to Theorem 1 for each value of k. The squared dots in Figure 2 represents the values of the upper bounds (shown on a scale in which we divide by k). We then simulated the index-based policy described in Section 4 for 10000 iterations respectively for $k = 2, 4, 8, 16$ respectively. The thinner lines in Figure 2 show 95% confidence intervals for the mean performance of this policy. As a baseline, we also simulate the equal allocation policy in which the m parallel computing resources are distributed equally to the k systems. The equal allocation policy is simulated for $50,000$ replications for each value of k, and 95% confidence intervals for the mean performance are shown as the thicker lines in Figure 2.

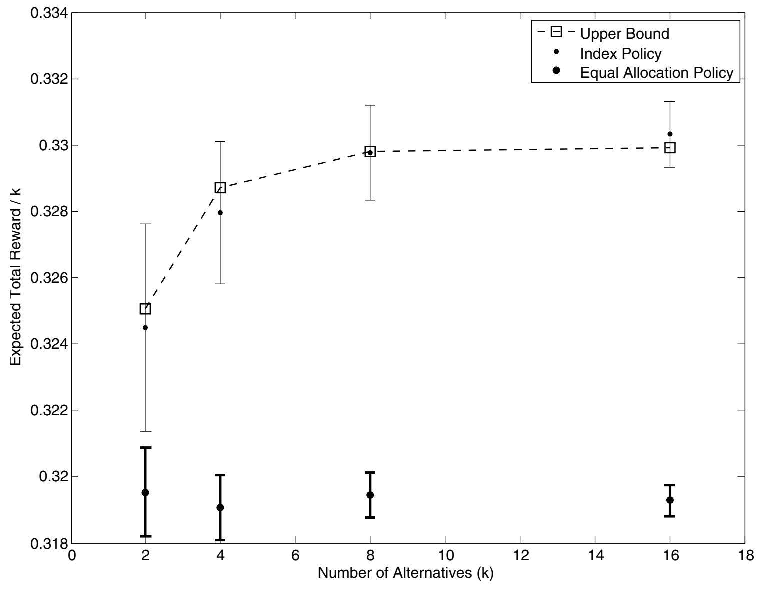


Figure 2: This figure shows the upper bound on the performance of the optimal policy for the MCS problem (dashed line with squares) normalized by dividing by k, as well as the estimated performance of two sub-optimal policies: the index policy from Section 4 (thinner lines and dots); and the equal allocation policy (thicker lines and dots). The setting pictured uses $m = k$, $d_x = 0.2$, $\alpha_{0x} = \beta_{0x} = 1$, $c_x = 0$. We use 10,000 independent replications to estimate the value of the index policy, and 50,000 for the equal allocation policy. The plot shows that the index policy is substantially better than equal allocation, and is statistically indistinguishable from optimal given the number of replications performed.

Confidence intervals for the index policy are wider than those for the equal allocation policy because fewer samples are taken when estimating the expected value of the index policy. This is because each simulation of the index policy takes a substantial amount of time in our current implementation. For the same reason, the index policy's confidence intervals still overlap the upper bounds. If we took more samples, we expect that the upper limit of the confidence interval would eventually fall below the upper bound, because we do not think that our policy is optimal. Nevertheless, our results show that the index policy performs substantially better than the equal allocation policy in the setting studied.

In the figure, the upper bound, divided by k, initially increases, and then levels out. This can be understood as follows. Because our initial state $S_{0,x}$ is identical for each system, our upper bound 12 can be rewritten as $\mathrm{UB}(\mathbf{S}_0) = \inf_{\boldsymbol{\lambda} \geq \mathbf{0}} \left[kV_{0,x}^{\boldsymbol{\lambda}}(S_{0,x}) + k\sum_{n=1}^{N} \lambda_n \right]$. where x is any arbitrary x. Dividing by k provides $\frac{1}{k}\mathrm{UB}(\mathbf{S}_0) = \inf_{\boldsymbol{\lambda} \geq \mathbf{0}} \left[V_{0,x}^{\boldsymbol{\lambda}}(S_{0,x}) + \sum_{n=1}^{N} \lambda_n \right]$.

$V_{0,x}^{\boldsymbol{\lambda}}$ does not depend on k directly, but it does depend on m which is the constraint on the maximum number of samples that can be taken from system x in any batch, and we set $m = k$ in our experiments. Thus, when we increase k, we loosen this constraint. We believe this is why $\frac{1}{k}\mathrm{UB}(\mathbf{S}_0)$ increases initially with k. Then, as k grows large, this constraint is no longer binding, as the optimal value of $\boldsymbol{\lambda}$ causes us to take less than $m = k$ samples in each batch. We believe this is why $\frac{1}{k}\mathrm{UB}(\mathbf{S}_0)$ levels off as k becomes large.

6 CONCLUSION

We offered a computationally feasible way to obtain the upper bound on the total expected reward of the finite-horizon MCS problem through Lagrangian relaxation. We then proposed an index-based policy using this Lagrangian relaxation. Using the upper bound as a reference, we showed this index policy performs close to optimal by running numerical experiments on a specific set of parameters.

ACKNOWLEDGMENTS

Peter Frazier was supported by NSF CAREER CMMI-1254298, NSF IIS-1247696, AFOSR YIP FA9550-11-1-0083, and AFOSR FA9550-12-1-0200.

REFERENCES

Andradóttir, S., and S. H. Kim. 2010. "Fully Sequential Procedures for Comparing Constrained Systems via Simulation". *Naval Research Logistics* 57 (5): 403–421.

Batur, D., and S. H. Kim. 2010. "Finding Feasible Systems in the Presence of Constraints on Multiple Performance Measures". *ACM Transactions on Modeling and Computer Simulation* 20 (3): 13:1–13:26.

Bofinger, E., and G. J. Lewis. 1992. "Two-Stage Procedures for Multiple Comparisons with a Control". *American Journal of Mathematical and Management Sciences* 12 (4): 253–275.

Chen, X., Q. Lin, and D. Zhou. 2013. "Optimistic Knowledge Gradient Policy for Optimal Budget Allocation in Crowdsourcing". In *Proceedings of the 30th International Conference on Machine Learning*, 64–72.

Damerdji, H., and M. K. Nakayama. 1996. "Two-Stage Procedures for Multiple Comparisons with a Control in Steady-State Simulations". In *Proceedings of the 1996 Winter Simulation Conference*, 372–375. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

DeGroot, M. H. 1970. *Optimal Statistical Decisions*. New York: McGraw Hill.

Dudewicz, E. J., and S. R. Dalal. 1983. "Multiple Comparisons with a Control When Variances Are Unknown and Unequal". *American Journal of Mathematics and Management Sciences* 3 (4): 275–295.

Dunnett, C. W. 1955. "A Multiple Comparison Procedure for Comparing Several Treatments with a Control". *Journal of the American Statistical Association* 50 (272): 1096–1121.

Dynkin, E. B., and A. A. Yushkevich. 1979. *Controlled Markov Processes*. New York: Springer.

Frazier, P. I. 2011. "Learning with Dynamic Programming". In *Wiley Encyclopedia of Operations Research and Management Science*, 1–13. Hoboken, New Jersey: Wiley.

Gittins, J., K. Glazebrook, and R. Weber. 2011. *Multi-armed Bandit Allocation Indices*. 2nd ed. Hoboken, New Jersey: Wiley.

Healey, C., S. Andradóttir, and S.-H. Kim. 2014. "Selection Procedures for Simulations with Multiple Constraints under Independent and Correlated Sampling". *ACM Transactions on Modeling and Computer Simulation* 24 (3): 14:1–14:25.

Kim, S. H. 2005. "Comparison with a Standard via Fully Sequential Procedures". *ACM Transactions on Modeling and Computer Simulation* 15 (2): 155–174.

Kim, S. H., and B. L. Nelson. 2007. "Recent Advances in Ranking and Selection". In *Proceedings of the 2007 Winter Simulation Conference*, 162–172. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Nelson, B. L., and D. Goldsman. 2001. "Comparisons with a Standard in Simulation Experiments". *Management Science* 47 (3): 449–463.

Paulson, E. 1952. "On the Comparison of Several Experimental Categories with a Control". *The Annals of Mathematical Statistics* 23 (2): 239–246.

Powell, W. B. 2007. *Approximate Dynamic Programming: Solving the curses of dimensionality*. New York: John Wiley and Sons.

Szechtman, R., and E. Yücesan. 2008. "A New Perspective on Feasibility Determination". In *Proceedings of the 2008 Winter Simulation Conference*, 273–280. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Whittle, P. 1988. "Restless bandits: Activity Allocation in a Changing World". *Journal of Applied Probability* 25:287–298.

Xie, J., and P. Frazier. 2013a. "Upper Bounds on the Bayes-Optimal Procedure for Ranking & Selection with Independent Normal Priors". In *Proceedings of the 2013 Winter Simulation Conference*, 877–887. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Xie, J., and P. I. Frazier. 2013b. "Sequential Bayes-Optimal Policies for Multiple Comparisons with a Known Standard". *Operations Research* 61 (3): 1174–1189.

AUTHOR BIOGRAPHIES

WEICI HU is a PhD student in the School of Operations Research and Information Engineering at Cornell University. She received a B.A. in mathematics from Smith College. She has a research interest in simulation optimization and Bayesian statistics. Her e-mail is wh343@cornell.edu.

PETER I. FRAZIER is an assistant professor in the School of Operations Research and Information Engineering at Cornell University, and received a Ph.D. in Operations Research and Financial Engineering from Princeton University in 2009. He is the recipient of an AFOSR Young Investigator Award and an NSF CAREER Award. He is an associate editor for Operations Research, ACM Transactions on Modeling and Computer Simulation and IIE Transactions. His research interest is in dynamic programming and Bayesian statistics, focusing on the optimal acquisition of information and sequential design of experiments. He works on applications in simulation, optimization, operations management, medicine, and materials science. His email address is pf98@cornell.edu and his web page is <www.orie.cornell.edu/pfrazier>.

JING XIE is a risk & information manager at American Express Company. She received a Ph.D. in Operations Research & Information Engineering from Cornell University in 2014. Her Ph.D. thesis was on Bayesian designs for sequential learning problems. Her e-mail is joyce.jingx@gmail.com and her web page is <http://people.orie.cornell.edu/jx66/>.